

A Heuristic Approach for On-line Discovery of Unidentified Spatial Clusters from Grid-based Streaming Algorithms

Marcos Roriz Junior¹, Markus Endler¹, Marco A. Casanova¹, H elio Lopes¹, and Francisco Silva e Silva²

¹ Department of Informatics - Pontifical Catholic University of Rio de Janeiro
{mroriz, endler, casanova, lopes}@inf.puc-rio.br

² Department of Informatics – Federal University of Maranh o
fssilva@deinf.ufma.br

Abstract. On-line spatial clustering of large position streams are useful for several applications, such as monitoring urban traffic and massive events. To rapidly and timely detect in real-time these spatial clusters, algorithms explored grid-based approaches, which segments the spatial domain into discrete cells. The primary benefit of this approach is that it switches the costly distance comparison of density-based algorithms to counting the number of moving objects mapped to each cell. However, during this process, the algorithm may fail to identify clusters of spatially and temporally close moving objects that get mapped to adjacent cells. To overcome this answer loss problem, we propose a density heuristic that is sensible to moving objects in adjacent cells. The heuristic further subdivides each cell into inner slots. Then, we calculate the density of a cell by adding the object count of the cell itself with the object count of the inner slots of its adjacent cells, using a weight function. To avoid collateral effects and detecting incorrect clusters, we apply the heuristic only to transient cells, that is, cells whose density are less than, but close to the density threshold value. We evaluate our approach using real-world datasets and explore how different transient thresholds and the number of inner slots influence the similarity and the number of detected, correct and incorrect, and undetected clusters when compared to the baseline result.

Keywords: Answer Loss On-line Spatial Clustering, On-line Spatial Clustering, On-line Grid-based Spatial Clustering, On-line Clustering

1 Introduction

Advances in mobile computing enabled the popularization of portable devices with Internet connectivity and location sensing [1]. Collectively, these devices can generate large position data streams, which can be explored by applications to extract patterns. A mobility pattern that is particularly relevant to many applications is clustering [2], a concentration of mobile devices (*moving objects*) in some area, *e.g.*, a mass street protest, a flash mob, a sports or music event, a traffic jam, etc.

A fast and on-line (continuous) detection of spatial clusters from position data streams is desirable in numerous applications [1, 3]. For example, for optimizing traffic flows (*e.g.*, rapidly detecting traffic jams), or ensuring safety (*e.g.* detection of suspicious or anomalous movements of rioters).

Traditionally, such clusters were detected using off-line techniques [3], where data were processed in batch (*e.g.*, from a dataset or a database). Most of these techniques are based on the classic DBSCAN algorithm [4, 5], which provides a density-based definition for clusters. DBSCAN uses two thresholds: an ε distance and the minimum density $minPts$ of moving objects to form a cluster. A moving object p that has more than $minPts$ moving objects in its ε -Neighborhood is a *core moving object*, where the ε -Neighborhood of p is the set $N_\varepsilon(p) = \{q \in D \mid distance(p, q) \leq \varepsilon\}$ and D is the set of all moving objects. The main idea of DBSCAN is to recursively visit each moving object in the neighborhood of a dense moving object to discover other dense objects. By such, a cluster is (recursively) expanded until no further objects are added to it.

However, to timely detect such clusters, the assumptions of traditional clustering algorithm can become troublesome. For example, to obtain the neighborhood set $N_\varepsilon(p)$ of a moving object p [6], DBSCAN compares the distance between p and the remainder moving object in order to select those that are within ε distance. Since these techniques were designed for off-line scenarios, they can employ spatial data structures, such as R-Trees and Quad-Trees, which provide efficient indexes for spatial data. However, for continuous-mode cluster detection in data streams, this optimization becomes troublesome, due to the high cost of continuously maintaining the spatial tree balanced. To overcome this issue, some approaches employ [5] incremental algorithms to reduce the number of distance comparisons. However, even these approaches become problematic in data stream scenarios due to the difficulty in accessing and modifying the spatial tree in parallel [7, 8], *e.g.*, to handle several data items at once. For example, to avoid inconsistencies in the tree index, these approaches sequentially execute the algorithm for each data item, which does not scale to large data streams (*e.g.*, with thousands of items per seconds).

Algorithms [8–11] that combine grid and density-based clustering techniques have been proposed as means to address such challenges. Such approaches handles this issue by dividing the monitored spatial domain into a grid G of $\frac{\varepsilon}{\sqrt{2}} \times \frac{\varepsilon}{\sqrt{2}}$ cell segments such that the maximum distance between any two moving objects inside a cell is ε . Then, rather than measuring the distance between each pair of moving objects, it counts the number of moving objects mapped to each cell. Cells that contain more than a given threshold ($minPts$) of moving objects are further clustered. This process triggers an expansion step that recursively merges a dense cell with its adjacent neighbor cells. Since cells are aligned in a grid, the recursive step is straightforward. With this approach, the main performance bottleneck is not anymore the distance comparison, but the number of grid cells. A primary benefit of grid-based algorithms is that they transform the problem semantics from distance comparison to counting.

Although the counting semantics enables grid-based approaches to scale and provides faster results over other approaches, they may fail to identify some clusters, a problem known as *answer loss* (or *blind spot*) [12–14]. Answer loss is a problem that happens in any grid-based approach, such as [8–11], due to the discrete subdivision of

the space domain into grid cells, which can lead to spatial and temporally close moving objects being mapped to different cells and, thus, not contributing for a cell density that exceeds the $minPts$ even though the objects are closer than ϵ to each other. For example, consider the clustering scenario shown in Fig. 1. Suppose that the threshold for a dense cell is $minPts = 4$. Then, although the moving objects are close to each other *w.r.t.* ϵ , the grid-based clustering process will not detect such cluster since none of the cells is considered dense.

To address the *answer loss* problem, in this paper we propose a counting density heuristic that is sensitive to the number of moving objects in adjacent cells. The idea is to further subdivide cells into “logical” slots and to consider the distribution of moving objects inside these slots on adjacent cells when calculating the cell density. We consider two discrete functions (linear and exponential) to weight the adjacent cells inner slot distributions and combine the result with the cell’s own density. However, since the heuristic considers moving objects in adjacent cells when calculating the cell density, it can wrongly detect a cluster that does not exist as a collateral effect. Thus, we propose that the heuristic should be used only when the evaluated cell has a *transient density*, *i.e.*, the number of its objects is less than the required $minPts$, but larger than a lower threshold. We evaluate the tradeoff between the transient cell threshold with the similarity and the number of detected and undetected clusters of a grid-based cluster result using the heuristic, when compared to the baseline (DBSCAN) off-line clustering algorithm. We also evaluate if and how the number of slots impacts the clustering result. Hence, the main contributions of this paper are:

- A counting heuristic, with linear and exponential weights, for mitigating the answer loss problem of on-line grid-based clustering approaches in spatial data streams;
- An extensive evaluation of the proposed heuristic using a real-world data stream.

The remainder of the paper is structured as follows. Section 2 overviews the concepts used throughout the paper. Section 3 presents our approach, the density heuristic that considers the moving objects in adjacent cells. Section 4 extensively evaluates and discusses the proposed technique using a real world dataset. Section 5 summarizes related work, while Section 6 states the concluding remarks and our plans for future work.

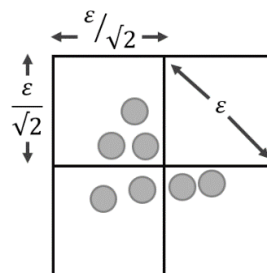


Fig. 1. Answer loss (blind spot) issue in grid-based clustering approaches ($minPts = 4$).

2 Basic Concepts

As mentioned before, on-line grid-based clustering algorithms [8–11] have been proposed as a means of scaling the clustering process [3]. The overall idea of these approaches is to cluster the grid cells, rather than moving objects, by counting the moving objects in each cell. Since cells are aligned in a grid, the expansion step is straightforward. To do this process, a spatial domain $S = ([lat_{min}, lat_{max}], [lng_{min}, lng_{max}])$, defined by a pair of latitude and longitude intervals, is divided into a grid into a grid G of $\frac{\varepsilon}{\sqrt{2}} \times \frac{\varepsilon}{\sqrt{2}}$ cells. As said, the choice for an $\frac{\varepsilon}{\sqrt{2}} \times \frac{\varepsilon}{\sqrt{2}}$ partition is to guarantee that the maximum distance between any moving object inside a cell is ε . Each cell G_{ij} contains objects in the latitude and longitude intervals respectively defined as:

$$(lat_{min} + j \times \frac{\varepsilon}{\sqrt{2}}, lat_{min} + (j + 1) \times \frac{\varepsilon}{\sqrt{2}}] \text{ and } (lng_{min} + i \times \frac{\varepsilon}{\sqrt{2}}, lng_{min} + (i + 1) \times \frac{\varepsilon}{\sqrt{2}}]$$

Then, the moving objects in the data stream are mapped to these cells. The number of moving objects mapped to a cell within a given period is called the *cell density*. Cells that have a density greater than or equal to the *minPts* threshold are further clustered (expanded). Due to the grid alignment, this expansion is simpler than the pairwise distance comparison in DBSCAN. The condition for a grid cell to be part of a cluster is to have a density higher than the *minPts* threshold or else to be a border cell of some dense cell, analogously to the core and border objects in DBSCAN. A cluster output is thus the set of cells, core and border, reached from the triggered dense cell.

However, the discrete space subdivision into grid cells can lead to answer loss (or blind spot) problems [12, 13, 15, 14]. Due to this division, spatially and temporally close moving objects may be mapped to adjacent cells. Although the moving objects are close *w.r.t.* ε , such cluster will not be detected since no cell is dense *w.r.t.* *minPts*.

To address this issue, we proposed a density heuristic function that considers moving object in adjacent cells when evaluating the *cell density*. The main goal of the heuristics is to discover and output clusters that would not be discovered using only the cell density. We aim at providing a clustering result as close as possible to that of DBSCAN, which we take as ground truth. We will show that our heuristics is capable of on-line detecting a higher number of clusters to the off-line DBSCAN ground-truth result.

3 Proposed Approach

To address the *answer loss* problem while retaining the clustering counting semantic, the proposed heuristic logically divide each adjacent cell into s inner slots, in both dimensions (horizontal and vertical). Then, the density function counts the number of mapped objects in those slots in a way that slots closer to G_{ij} have higher weight than those that are more distant. The exact process is detailed below.

First, each cell is logically divided into vertical and horizontal inner slots (strips) of equal width and height. Each object mapped to a cell G_{ij} is further associated with one of these slots. This operation can be done in constant time during the cell mapping step of a grid algorithm by comparing the location update position with the size of each slot.

The next step is to update the clustering density function by considering the inner density of the neighboring cells, as illustrated by Fig. 2. Not only the number of mapped objects in a cell G_{ij} is considered, but also the distribution of moving objects in the inner slots of each neighboring cell in $N_\varepsilon(G_{ij})$. To do this, we propose a discrete decay weight function that counts the number of moving objects inside each of the inner slots of each neighbor cell in such way that slots closer to G_{ij} receive a higher weight. The closer slots indexes vary according to the position of the neighboring cell, as shown by darker tones in Fig. 2. Thus, to avoid such specific cases when computing the density function, we “normalize” the neighboring cells inner slots distribution. To do so, we reorder the neighbor cells $\rho \in N_\varepsilon(G_{ij})$ inner slots $n_{\rho,i}$ in such a way that the first slot, $i = 0$, is closer to the evaluated cell, while the last slot, $i = s - 1$, is the farthest, to enable them to be handled as if they were aligned in the same position.

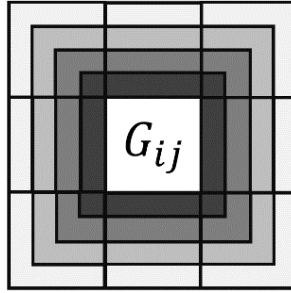


Fig. 2. Density neighborhood of a given cell. Note that the neighbor’s closer inner slots vary according to the position of the neighbor.

After normalization, the density function can be described as:

$$density(G_{ij}) = |G_{ij}| + \sum_{\rho \in N_\varepsilon(G_{ij})} (\sum_{i=0}^{s-1} w_i \times n_{\rho,i})$$

where $|G_{ij}|$ is the number of moving objects contained in G_{ij} , $\rho \in N_\varepsilon(G_{ij})$ is a cell adjacent neighbor, s is the number of inner slots, $n_{\rho,i}$ is the number of moving objects in slot i index of neighboring cell ρ , and w_i is the i^{th} decay weight.

We propose two discrete weight decay functions, a linear and an exponential one, illustrated by Fig. 3 (for $s = 4$ inner slots). The linear decay weights can be computed as $w_i = \frac{-i}{s} + 1$, where i is the given cell inner slot index, *e.g.*, considering $s = 4$ as inner slots, the slots weights are $w_0 = 1$, $w_1 = 0.75$, $w_2 = 0.50$, and $w_3 = 0.25$. The exponential decay weights can be computed as $w_i = k^i$, where k is a number between 0 and 1 such that $k^s \cong 0$. Based on this definition, k varies accordingly to the number of inner slots s . For example, considering that cells have $s = 4$ inner slots, k value is approximately 0.3162, *i.e.*, $0.3162^4 \approx 0$, while for grid cells that have $s = 10$ inner slots, k is approximately 0.6309, since $0.6309^{10} \approx 0$. To discover k , one can assume $k^s = 0.01$, then $\ln k^s = \ln 0.01$, which yields $k = e^{\frac{\ln 0.01}{s}}$. For example, as said, for $s = 4$, k is approximately 0.3162 and the slot weights are $w_0 = 1$, $w_1 \approx 0.3162$, $w_2 \approx 0.3162^2 \approx 0.099$ and $w_3 \approx 0.3162^3 \approx 0.0312$.

By applying the discrete weight function to the neighboring cells inner slots, the proposed heuristic can detect several answer loss clustering scenarios using a counting semantics. For example, consider the clustering scenario of Fig. 4 (a) and parameters $s = 4$ and $minPts = 4$. Since the cell density is 2, the standard grid algorithm would not detect the cluster. Using the proposed heuristic, with a linear weight decay, the computed density will be $2 + (1 \times 1) + (4 \times 0.75) = 5 \geq minPts$, thus, the cluster would be detected. An exponential decay weight will also detect this cluster, since the computed density would be $2 + (1 \times 0.3162^0) + (4 \times 0.3162^1) = 4.26 \geq minPts$.

On the other hand, as a collateral effect of considering moving objects of neighboring cells when calculating the cell density, the proposed heuristic would detect a non-existing cluster (a false positive) in some situations, as illustrated in the cell configuration of Fig. 4 (b), for $s = 4$ and $minPts = 4$. In this scenario, the standard grid algorithm would correctly not detect the cluster, since the cell density is 1. However, the linear weight decay would wrongly detect the cluster, since the cell density in this case would be $1 + (1 \times 1) + (1 \times 0.75) + (2 \times 0.5) + (1 \times 0.25) = 4 \geq minPts$. Nevertheless, in this scenario, the exponential weight decay would correctly not detect such cluster, since the computed density would be $1 + (1 \times 0.3162^0) + (1 \times 0.3162^1) + (2 \times 0.3162^2) + (1 \times 0.3162^3) = 2.54 \leq minPts$.

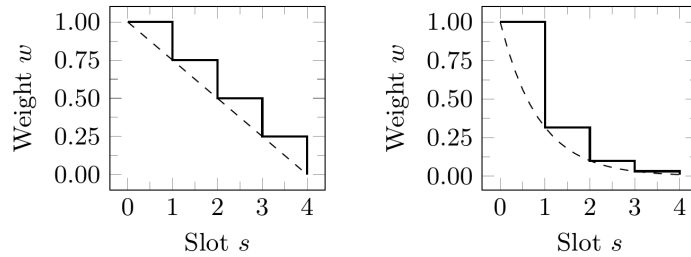


Fig. 3. Linear and exponential discrete weight for the density function (for $s = 4$ inner slots).

To mitigate the heuristic collateral effect of detecting non-existing clusters, we propose to only apply the method when evaluating *transient cells*, that is, cells whose density are lower than $minPts$, but higher than a lower-bound $lowerPts$ threshold, where $lowerPts \leq minPts$. By using a lower-bound threshold, we can restrict the heuristic application to specific scenarios, *e.g.*, to almost dense cells. For example, consider a $lowerPts = 2$ threshold in the clustering scenario of Fig. 4 (b). In this configuration, the linear weight heuristic will correctly not detect the cluster, since the cell density would be 2. In addition to mitigating false positive answers, transient cells also reduce the overall computational cost, since the heuristic will only apply to cells whose density are within the interval $lowerPts \leq minPts$. For example, a $lowerPts = 0$ means that the heuristic is applied to every updated moving objects. Although the heuristic employs a counting semantic, for large data streams, this can increase the computation cost and delay the cluster discovery. However, using a high $lowerPts$ threshold value may again raise the answer loss problem, since the heuristic would only apply to values close to $minPts$. For example, in Fig. 4 (a), for $lowerPts = 3$, the cluster would not be detected, since the heuristic density would not be applied.

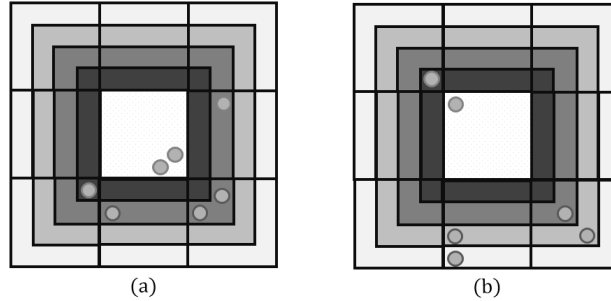


Fig. 4. Cell configurations scenarios. In (a) the scenario forms a cluster, while in (b) it does not.

4 Evaluation

The proposed heuristic was evaluated using real-world data streams and DG2CEP [9], a grid-based on-line clustering algorithm for spatial data streams. Using the heuristic-enhanced DG2CEP, the evaluation had two goals:

- Measure the percentage and similarity of the on-line cluster results found, when compared with the original DG2CEP and the baseline DBSCAN off-line algorithm. Furthermore, investigate how these results vary with different *lowerPts* values that define transient cells.
- Investigate if the number of correct and incorrect clusters found and their similarity with the baseline algorithm vary when using different number of inner slots s .

To do so, first, we used a dataset from the bus fleet data stream of the city of Rio de Janeiro, in Brazil, and computed a second-by-second off-line DBSCAN algorithm over this dataset. This second-by-second off-line clustering output is used as the ground-truth result, which enables us to compare the on-line clustering result when using the proposed heuristic. By comparing the outputs, in addition to the cluster similarity (*e.g.*, precision), we can measure the number of detected and undetected (missed) clusters.

4.1 Implementation

Since we aim at discovering the spatial clusters in real-time, we implemented the proposed heuristic as an module in DG2CEP [9], an on-line grid-based spatial clustering algorithm which uses the counting semantic to cluster in real-time streams of spatial data. We opted to implement the heuristic in DG2CEP due to its on-line processing flow, rather than operating in batch as other grid-based algorithms do [8, 10]. To process on-line streams and provide real-time results, DG2CEP uses Complex Event Processing (CEP) concepts [16]. CEP provides a set of data stream concepts based on continuous queries and event-condition-action rules, such as filtering and enrichment. Each continuous operation is performed by a processing stage, named Event Processing Agent (EPA). DG2CEP defines a network of such stages (EPAs) to enable a continuous detection of the formation and dispersion of spatial clusters.

Both the heuristic-enhanced and the original DG2CEP algorithms were implemented in the Java programming language and made use of the popular open source Esper CEP engine [17]. The heuristic uses several CEP processing stages (EPAs), such as filter and projection, to identify and extract cells that are within the *lowerPts* transient threshold. In addition, when a location update is mapped to a different cell than a previous one, we recheck the cell density with the heuristic to verify if such cell are still dense *w.r.t. minPts*.

4.2 Input Data Stream

The real world dataset is based on the bus fleet of the city of Rio de Janeiro, Brazil. We crawled the dataset¹ from the data.rio open platform, which contains trajectory data for the city 11,324 buses, with an average location update interval of 60 seconds. It is delimited by the following latitude and longitude intervals: $[-23.0693, -22.7664]$ and $[-43.7849, -43.1429]$, and contains the data of a single day, July 24th, 2014. We choose this day since it was the day with the highest number of updates over the week period.

As mentioned, to evaluate the number of discovered clusters, correct and incorrect, and the results similarity of the proposed heuristic over an on-line data stream, we computed the second-by-second ground-truth cluster result using the off-line DBSCAN clustering algorithm. To do so, at every second, we execute a DBSCAN over the dataset, considering the location updates in the 60 seconds time window, and store the clustering result as ground-truth information. Since this is a costly task, we opted for restricting this analysis to a two hour rush period of the dataset, from 17:00–19:00, given the probability of having more clusters due to traffic jams of the rush-hour traffic load.

4.3 Measurement Metrics

Whenever the proposed heuristic discovers a cluster we take a snapshot of its moving objects to analyze at a later time. Using this information, we compare the discovered clusters with their counterparts in the ground-truth log. A cluster c is discovered if, in the ground-truth snapshot, there exists a cluster d such that the overlap between c and d is larger than 50%, that is, the discovered cluster in the heuristic enhanced DG2CEP contains at least 50% of the moving objects of the ground-truth result. If the heuristic wrongly detects a cluster, *i.e.*, no similar cluster exists in the ground-truth log, then this detected cluster is marked as false positive (*FP*). All clusters not detected by the heuristic but present in the ground-truth log are marked as false negatives (*FN*). By comparing these metrics, the percentage of incorrectly detected clusters (*FP*) and missed clusters (*FN*), to the total number of clusters in the ground-truth log, we can measure the heuristic’s effectiveness of handling the answer loss problem.

¹ The crawled dataset is available to be downloaded and reproduced at: <http://www.lac.inf.puc-rio.br/answerloss/>.

To understand the quality of detected clusters, after detecting a cluster we measure its similarity with its DBSCAN counterpart using the F_1 metric. F_1 is a accuracy metric that considers the precision and recall between the detected cluster and its ground-truth counterpart. Its values range from 0 to 1, such that the closer the value is to 1 the more accurate the results are. Precisely, when F_1 is 1 the cluster contain the same moving objects, while 0 means that they are totally different (no common objects). It is defined as $F_1 = 2 \times \frac{precision \times recall}{precision + recall}$, where $precision = \frac{tp}{tp+fp}$ and $recall = \frac{tp}{tp+fn}$, considering that tp is the number of pairs of moving objects in both clusters, fp as the number of pairs of moving objects that is in the detect cluster but not in the ground-truth result, and fn the number of pairs of moving objects that is in the ground-truth result but not in the detected cluster.

4.4 Parameters of the Experiment

We choose the parameters of the experiment based on a hypothetical application for detecting traffic jams or massive events caused by a concentration of buses in a given region. In this scenario, the spatial clusters are formed when a set of different buses that are close to each other exceeds a given size threshold. Such application can be useful to dynamically control the number of buses serving the citizens or, for example, to alert other drivers to avoid the specified region.

Based on the application scenario and the data stream frequency, we set the window size to be $\Delta = 60$ seconds, that is, we consider the location updates received within the last 60 seconds. Furthermore, we choose two grid sizes, $\epsilon = 100$ m and $\epsilon = 200$ m, for the respective density thresholds $minPts = 20$ and $minPts = 40$.

To measure the impact of the proposed heuristic, we considered *lowerPts* thresholds ranging from 90% to 30% of the *minPts* density threshold. For example, for $minPts = 20$, we evaluate the following *lowerPts* thresholds: 18, 16, 14, 12, 10, 8, and 6, while for $minPts = 40$ we considered the following *lowerPts* thresholds: 36, 32, 28, 26, 24, 20, 16, and 12. Finally, we choose the number of slots to be $s = 10$. Thus, for $\epsilon = 100$ m, each inner slot width is approximately 10 meters, while for $\epsilon = 200$ m, it is 20 meters. We choose these values considering that the GPS accuracy is approximately between 10 to 20 meters. However, in this evaluation, we also investigated if the number of slots s impact the number and similarity of the detected clusters. For this test, we considered the following number of slots: 10, 50, and 100.

4.5 Experiment Setup

We executed all experiments in the Microsoft[®] Azure Cloud, using two virtual machines running Ubuntu GNU/Linux 14.04.3 64-bit and the OpenJDK 1.7.91 64-bit Java runtime. The first machine contains a DG2CEP deployment, while the second one contains a dataset playback tool. The virtual machines were interconnected through a Gigabit link/bus and had the following hardware configuration:

- 4 Cores Intel[®] Xeon CPU E5-2673 v3 @ 2.40 GHz
- 14 GiB Memory RAM

4.6 Result and Analysis

In this subsection, we present the evaluation results. Each experiment was run 10 times and the error bars in the graphs represent a 99% confidence interval. Each test replays a two-hour dataset, 17:00–19:00, to the input stream of the on-line clustering algorithm.

Fig. 5 illustrates the percentage of undetected clusters (false negative - FN) and incorrectly detected clusters (false positive - FP) of the total clusters found by the heuristic enhanced DG2CEP when compared with DBSCAN's ground-truth results, for parameters $\varepsilon = 100$, $minPts = 20$, and $s = 10$. It also illustrates how each of these metrics and the similarity (F_1) of the set of detected clusters vary when using different $lowerPts$ thresholds, where the $lowerPts$ threshold is represented as a percentage of the original $minPts$ thresholds. Subfigure (a) and (b) of represents the values obtained when evaluating the heuristic with linear and exponential weights respectively.

As expected, in both cases (linear and exponential), due to the *answer loss* problem, the graph shows that when $lowerPts$ is equal or close to $minPts$ (100% and 90%) the algorithm will fail to detect the majority of the clusters (FN). However, since the threshold is closer to $minPts$, the original density threshold, there will be few collateral effects, hence, the low percentage of incorrect detected clusters (FP).

According to the linear weight graph, Fig. 5 (a), the $lowerPts$ thresholds that yielded the best tradeoff results were 60% and 50% of $minPts$. These thresholds reduced the number of undetected clusters from 80% to 23.57% and 15.32%, respectively, with a collateral effect of incorrect clusters of 13.51% and 19.05%, respectively. Interestingly, in the heuristic, using both weights (linear and exponential), the detected clusters of all thresholds presented a high precision and recall (F_1) of over 90%, with its ground-truth counterpart.

The heuristic exponential weight graph, Fig. 5 (b), presented better results as $lowerPts$ decreases. This illustrates that, when using the exponential weight, the heuristic is more tolerant to collateral effects. For example, the number of incorrect clusters (false negative) results is 4.47% for a $lowerPts$ equal to 30% of $minPts$. However, for this parameter, the heuristic reduced the number of cluster not detected due to the answer loss problem, from 80% to 31.51%, instead to 13.51% when using linear weights.

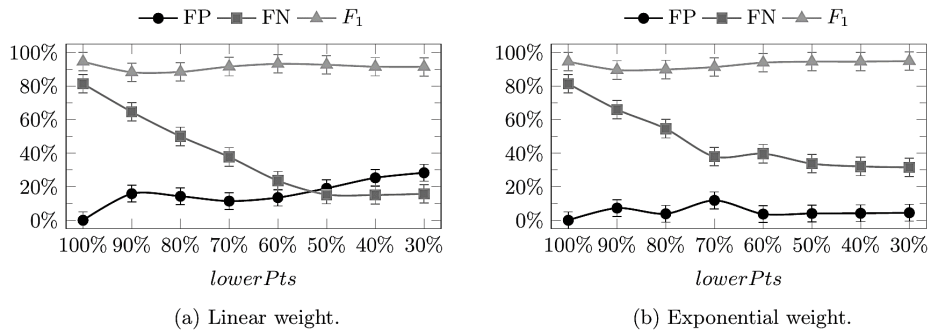


Fig. 5. Evaluation result of the proposed heuristic for $\varepsilon = 100$ and $minPts = 20$.

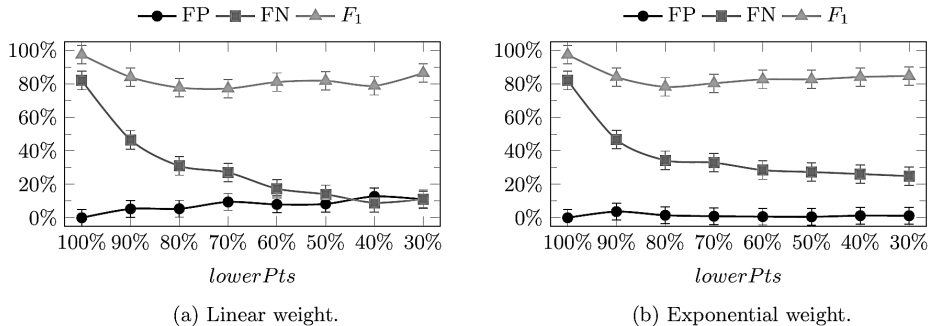


Fig. 6. Evaluation result of the proposed heuristic for $\epsilon = 200$ and $\text{minPts} = 40$.

Similarly to Fig. 5, Fig. 6 illustrates the number of FN (undetected clusters), FP (incorrectly detected) and the similarity of the clusters detected, for parameters $\epsilon = 200$, $\text{minPts} = 40$, and $s = 10$. The graph results are similar to those obtained for $\epsilon = 100$ and $\text{minPts} = 20$. The major difference is that, for both weight functions, the detected clusters similarity (F_1) with their ground-truth counterpart decreases to an average of 80%, while the clusters from the previous experiment had over 90% of similarity, for all parameters configuration. The primary reasons for this reduction comes from having a bigger cell ϵ width. Indeed, remember that a grid cluster is a combination of dense cells and their borders cells, *i.e.*, in addition to the moving objects inside the dense cells, all border cells' moving objects are also included. Thus, a bigger cell width can potentially include a higher number of objects that are not part of the cluster.

Another difference between this experiment and the previous one is that, for the linear weight heuristic, Fig. 6 (a), the graph presents a reduction of approximately half of the collateral effects, *i.e.*, the number of wrongly detected clusters. More precisely, the number of false positive results dropped from 13.51% to 7.97%, for a lowerPts threshold equal to 60% of minPts . As the cell width increases, the slot size also increases, and it can lead to a better representation of the objects distribution.

With respect to the relationship between the number of cell divisions and the heuristic results, Fig. 7 and Fig. 8 shows how the number of undetected clusters (false negative) and incorrectly detected clusters (false positive) vary for different number of cell slots when using a linear and exponential weight respectively for parameters $\epsilon = 100$ and $\text{minPts} = 20$. Overall, the results presented a similar behavior. Intuitively, we expected that the heuristic would present a better result as the number of slots increased. However, this was not the case. It seems that the high number of cell slots combined with the GPS error failed to grasp the correct distribution of objects within the cell.

5 Related Work

In this section, we compare the proposed heuristic with other solutions that address the answer loss problem. In general, these approaches are aimed at off-line scenarios and extensively use spatial index and operators, contrarily to the counting semantic.

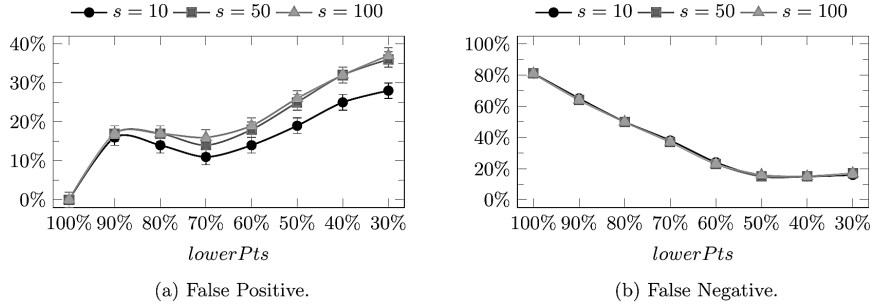


Fig. 7. Relationship between heuristic results and number of cell slots s , for linear weights

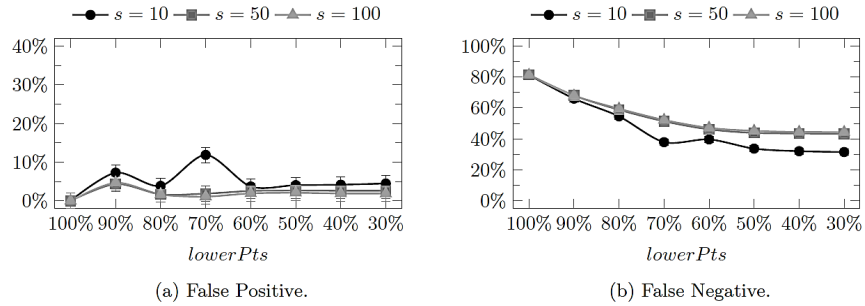


Fig. 8. Relationship between heuristic results and number of cell slots s , for exponential weights.

Ni *et al.* [13] presented two techniques, an exact and an approximate method that solve the *answer loss* problem in spatial dense queries. Both methods rely on a spatial index TPR-Tree, an R*-tree variant, to index the moving object trajectories. Furthermore, they extended the cell density concept to dense points, as if each point in the cell had its own ε -Neighborhood radius. The first method uses a filtering and refinement strategy. Cells that contain less than $minPts$ moving objects, but when combined with their neighborhood cells surpass this threshold, are filtered to be further analyzed. The refinement step applies a detailed plane-sweep algorithm to count the number of moving objects in the cell's neighborhood. To do so, it executes a sequence of spatial-temporal range queries in the TPR-tree. Then, by combining the queries answers, they are able to discover the moving objects that are within the cell radius. Although interesting, the approach requires a sequence of range-queries and spatial index operations, during the plane-sweep step, which can be troublesome to guarantee in data streams scenarios that requires timely responses. Hence, this approach is better suited to off-line scenarios where response time is not the primary concern, but correctness.

Their second technique is an approximation method that provides a function to represent the density distribution of moving objects. Contrary to their first method, this function represents the entire grid density, rather than considering each individual cells. More precisely, the function returns all dense regions (clusters) of the grid, *w.r.t.* $minPts$, in a given timestamp. The function is based on Chebyshev polynomials, a recursive function that uses several geometric primitives such as *cos* and *arccos*.

While using a single function to discover any dense cluster is appealing, the computational cost of such function is higher than counting the number of moving objects. In addition, the function can cause an overhead or delay the response since it calculates all dense regions at once, even those that do not suffer from the *answer loss* problem.

Jeung *et al.* [14] proposed an interesting supervised solution, which combines off-line processing and hidden Markov chains, for solving the *answer loss* problem in trajectory clustering. Their overall goal is to extract clusters from trajectories datasets. To do so, first, they preprocess a trajectory dataset using DBSCAN to discover the clusters location and their moving objects. Using the clustering result and hidden Markov chains, they create a trajectory model by discovering the set of cells and their probability to be associated with the cluster location, where the probability of each cells expresses the percentage of moving objects of such cluster in that cell. Based on this model, they can correlate the clusters location to the respective moving object cells. There are two main differences between their approach and ours. First, our proposed heuristic does not need to do any *a priori* processing. Second, their approach is focused on clusters solely from the *a priori* trajectory dataset, while our approach can dynamically discover and detect clusters from data streams.

6 Conclusion

This paper presented a density heuristic for on-line grid-based spatial clustering algorithms that addresses the *answer loss* problem, which occurs when clusters are not detected due to spatially and temporally close moving objects being mapped to adjacent cells. The heuristic provides a density function that is sensible to moving objects in adjacent cells while retaining the counting semantics of on-line grid-based approaches. To do so, we further divide each cell into horizontal and vertical slots. Then, we count the distribution of moving objects inside the slots of the adjacent cells when computing a cell density. Moreover, we proposed two discrete weight functions, a linear and an exponential, so that closer inner slots to the cell have higher weight than distant ones. To avoid collateral effects of wrongly detecting a cluster that does not exist, due to considering moving objects in adjacent cells, we proposed that the heuristic should be applied only when the evaluated cell has a *transient density*, *i.e.*, less than the required *minPts* threshold, but higher than a *lowerPts* threshold.

To investigate the relationship between the number of undetected clusters discovered and wrongly detected clusters, we evaluated the heuristic with several transient thresholds using real-world data streams and the on-line DG2CEP grid-cluster algorithm [9]. Based on the experiments, we concluded that, for linear weights, a *lowerPts* threshold between 60% to 50% of the original *minPts* threshold produced the best results. Specifically, for a *transient* threshold of 60%, the number of undetected clusters reduced from 80% to 23.57%, with a collateral effect of incorrect clusters of 13.51%. For exponential weights, we concluded that as *lowerPts* decreases *w.r.t.* *minPts*, and the number of undetected and incorrect clusters also decreased. Finally, although exponential weights are resilient to collateral effects, *e.g.*, 4.47%, the number of undetected clusters discovered was lower, 31.51%, instead of 13.51% when using linear weights.

As future work, we intend to investigate a hybrid weight function, which combines the benefits of linear and resilience of exponential weights. In addition, we are interested in exploring our heuristic with different datasets and other grid-based algorithms.

Acknowledgment

This work was partly funded by CNPq under grants 557128/2009-9, 303332/2013-1, 442338/2014-7, by FAPERJ under grants E-26-170028/2008, E-26/201.337/2014 and E-01/209996/2015, and by Microsoft Research.

References

1. Zheng, Y., Capra, L., Wolfson, O., Yang, H.: Urban Computing. *ACM Trans. Intell. Syst. Technol.* 5, 1–55 (2014).
2. Dodge, S., Weibel, R., Lautenschütz, A.-K.: Towards a Taxonomy of Movement Patterns. *Inf. Vis.* 7, 240–252 (2008).
3. Amini, A., Wah, T., Saboohi, H.: On Density-Based Data Streams Clustering Algorithms: A Survey. *J. Comput. Sci. Technol.* 29, 116–141 (2014).
4. Ester, M., Kriegel, H., Sander, J., Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: *Proc. 2nd Int'l. Conf. on Knowledge Discovery and Data Mining*. pp. 226–231 (1996).
5. Ester, M., Kriegel, H.-P., Sander, J., Wimmer, M., Xu, X.: Incremental Clustering for Mining in a Data Warehousing Environment. In: *Proc. of the 24rd Int'l. Conf. on Very Large Data Bases*. pp. 323–333. San Francisco, CA, USA (1998).
6. Han, J., Kamber, M., Pei, J.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2011).
7. Garofalakis, M., Gehrke, J., Rastogi, R.: Querying and Mining Data Streams: You Only Get One Look a Tutorial. In: *Proc. 2002 ACM SIGMOD Int'l. Conf. on Management of Data*. p. 635. ACM, New York, NY, USA (2002).
8. He, Y., Tan, H., Luo, W., Mao, H., Ma, D., Feng, S., Fan, J.: MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. In: *Parallel and Distributed Systems, 2011 IEEE 17th Int'l. Conf. on*. pp. 473–480 (2011).
9. Roriz Junior, M., Endler, M., da Silva e Silva, F.J.: An on-line algorithm for cluster detection of mobile nodes through complex event processing. *Information Systems*. (2016).
10. Chen, Y., Tu, L.: Density-based Clustering for Real-time Stream Data. In: *Proc. 13th ACM SIGKDD Int'l. Conf. on Knowledge Discovery and Data Mining*. pp. 133–142. USA (2007).
11. Jensen, C.S., Lin, D., Ooi, B.C.: Continuous Clustering of Moving Objects. *Knowl. Data Eng. IEEE Trans.* 19, 1161–1174 (2007).
12. Jensen, C.S., Lin, D., Beng Chin Ooi, Rui Zhang: Effective Density Queries on Continuously Moving Objects. In: *Proc. IEEE 22nd Int'l. Conf. on Data Engineering*, (2006).
13. Ni, J., Ravishankar, C. V.: Pointwise-Dense Region Queries in Spatio-temporal Databases. In: *Proc. IEEE 23rd Int'l. Conf. on Data Eng.*, pp. 1066–1075 (2007).
14. Jeung, H., Shen, H.T., Zhou, X.: Mining Trajectory Patterns Using Hidden Markov Models. In: *Data Warehousing and Knowledge Discovery*, Springer Berlin Heidelberg, (2007).
15. Etzion, O., Niblett, P.: *Event Processing in Action*. Manning Publications Co., USA (2010).
16. EsperTech: Esper - Complex Event Processing, <http://www.espertech.com/esper/>.